UCRL-WEB-200040

CHAOS: Linux from Livermore

Table of Contents

Preface

Scope: This manual explains the goals and user-relevant features of CHAOS, which is Livermore Computing's Clustered High-Availability Operating System (a locally modified version of Linux with associated system support software). This document explains the goals of the CHAOS project, then introduces the local modifications of the operating-system kernel, the collection of system-administration tools tuned to local cluster-management needs, and the other broad aspects of CHAOS service to users (such as file systems and resource management).

For a comparison of local Linux features with features of other LC operating systems (especially AIX), see Linux Differences (URL: http://www.llnl.gov/LCdocs/linux). For details on the user tools provided by the Simple Linux Utility for Resource Management (SLURM), see LC's SLURM Reference Manual (URL: http://www.llnl.gov/LCdocs/slurm).

Availability: Some version of CHAOS now runs on every LC Linux cluster, open and secure.

Consultant: For help contact the LC customer service and support hotline at 925-422-4531 (open e-mail: lc-hotline@llnl.gov, SCF e-mail: lc-hotline@pop.llnl.gov).

Printing: The print file for this document can be found at:

```
OCF: http://www.llnl.gov/LCdocs/chaos/chaos.pdf
SCF: https://lc.llnl.gov/LCdocs/chaos/chaos_scf.pdf
```

Introduction

CHAOS Goals

CHAOS (Clustered High Availability Operating System) is Livermore Computing's localized version of Linux. CHAOS is maintained by local software developers to meet the special needs of local users (and their system administrators).

WHAT.
The commercial Red Hat (brand) "boxed set" distribution of Linux forms the core of CHAOS. LC staff members have:

- *modified* some features to better support the scientific computing priorities typical at Livermore, and different from most business-oriented installations of Linux,
- *added* extra support for LC's very large clusters of nodes intended for parallel computations, and
- *focused* on just the hardware and software found in LC production systems, to maximize the relevant return on local programming investment.

WHY.
After experimenting with several approaches to Linux use on LC machines, the staff of the Livermore Linux project decided that the best way to provide a good production and program-development environment with Linux was to focus on issues usually neglected when Linux is sold, installed, or applied commercially. Central to this "Livermore model" of computing is an emphasis on:

- High-performance computing (HPC) techniques, including the use of large clusters, big data files, and very long-running jobs often seen at LC.
- LC computational needs, especially for systems that enable rich simulations of a narrow set of numerical problems assisted by extensive and carefully tailored technical support.
- Local system administration style (not superficial or turnkey but "deep," with strong, on-going feedback between system administrators and system-software developers).

In many ways this approach to Linux establishes once again a style of operating-system design and management common at Livermore during the LTSS/CTSS era a decade ago (when another locally developed system embodied commitment to the same three dominant characteristics listed here).

HOW.
CHAOS, with the local modifications and threefold emphasis described here, evolved from several years of LC collaborations with current or former vendors on experimental high-performance computing systems. Frustration with the pace or outcome of several of those efforts led LC to refocus on a more independent Linux path during 2001. The first large-scale deployments of CHAOS appeared in 2002 (on what was formerly called the secure-network Production Capacity Resource (clusters Adelie and Emperor), then on the 1152-node, open-network Multiprogrammatic Capability Resource cluster MCR).

LC's primary *administrative* strategies to build and refine CHAOS involve:

- a few carefully chosen company partnerships, rather than reliance on broad, amorphous, sometimes divisive Linux collaborative work groups or committees, and

- open source sharing of nonproprietary new features (under the DOE-approved GNU General Public License), rather than reliance on the usual UC/LLNL highly constrained approach to intellectual property management.

TECHNICAL BACKGROUND.
Additional technical information on CHAOS and related hardware (Intel-chip Linux cluster) trends is collected and posted by project developers on the OCF web site

```
http://www.llnl.gov/linux
```

as the CHAOS staff releases it to the computer-science community. Details on the CHAOS resource manager (called SLURM) appear in the separate SLURM Reference Manual (URL: http://www.llnl.gov/LCdocs/slurm) (a section below (page 14) summarizes SLURM's innovative user-support features). The "exec-shield" security feature (page 31) was added to CHAOS as version 3.0 gradually deployed on LC machines in late 2005.

CHAOS Features

Livermore's Linux project has addressed four specific technical problems, identified in 1998, that needed to be solved to meet the general computing goals described in the subsection above. CHAOS is the outcome of that project. CHAOS solves these four key problems by enriching open-source Linux with either current or still-unfolding special, locally developed, system features (each explained in its own section below). In this way it overcomes the lack of integration testing and release discipline that often undermines large-scale open-source collaboration.

Problems Addressed

The four problems that the distinctive extra features of CHAOS address are:

- A high-performance interconnect (page 10) (internal network or "switch") for message passing among Linux-based compute nodes.
  SOLUTION: device drivers for Quadrics QsNet.
- A scalable parallel file system (page 12) with both hardware and software support for fast parallel I/O.
  SOLUTION: Lustre Lite (software) and Blue Arc servers (hardware).
  See the Lustre section of LC's I/O Guide (URL: http://www.llnl.gov/LCdocs/ioguide/index.jsp?show=s7) for details.
- A portable (vendor-independent) resource manager (page 14) for batch jobs, optimized for LC's job-control needs and able to invoke any of several different job schedulers (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s3.4) (FIFO, backfill, or others).
  SOLUTION: SLURM, deployed in fall, 2003.
- Reliable and efficient administrative tools (page 17) for very large clusters of Linux-based compute nodes.
  SOLUTION: a family of CHAOS management tools, some for staff and some for users.

The CHAOS operating system kernel is based on commercial Red Hat kernel releases instead of generic "stable Linux kernel releases" to provide a (more) reliable, predictable, focused way to both fix errors and report new-found problems. LC has modified the Red Hat kernel to support the special computing needs of LLNL users with (a) added or updated device drivers, (b) increased resource limits (for example, CHAOS allows 8192 instead of just 1024 file descriptors per process), (c) crash dump support to manage trouble during big production runs, (d) serial console logging, and (e) changes to enable parallel debugging tools like TotalView.

To determine which version of CHAOS an LC machine is currently running, execute

```
uname -a
```

and look in the *third* field (from the left) in the output line returned. If that field contains

- [Red Hat] 2.4.21, then the machine uses CHAOS 2.0;
- [Red Hat] 2.6.9, then the machine uses CHAOS 3.0.

Diskless Node Support

Starting with CHAOS 3.2 (spring, 2007), this operating system supports compute clusters (such as Atlas, Zeus, Rhea, Hopi, and Yana) whose nodes have no local hard disks. The file systems that normally reside on each local disk are served instead from NFS-mounted remote disks.

Enabling diskless clusters has both advantages and disadvantages for users.

ADVANTAGES.
The prime user benefit of diskless clusters is reliability. With no local disks to fail on thousands of separate nodes, the cluster is up more often and repair costs are reduced.

DISADVANTAGES.
(1) Diskless nodes have no swap space. An application that runs out of memory (usually 16 Gbyte/node) cannot swap to disk. Normally the CHAOS Out Of Memory (OOM) killer terminates such applications (unless memory overcommit is disabled). SLURM jobs ended by the OOM killer receive a characteristic message in STDERR, with the format

```
       slurmd[host]: taskn: [name] terminated by OOOM killer
       slurmd[host]: task0: VmSize: xxxM RSS: yyyM
```

where

host	is the node where the killed task was running,
task*n*	is the task ID of the terminated task,
name	specifies the process that was killed,
xxx	is the virtual memory size of the killed process, and
yyy	is the resident size of that process when it was killed.

(2) Temporary file systems use RAM, not disk. This means that any files in /tmp or /var/tmp use real memory on the node, rather than disk space. If you delete files from these temporary file systems, CHAOS reclaims the memory used. Also, CHAOS purges these file systems completely between jobs on diskless nodes. To preserve files between jobs, therefore, you must use HPSS archival storage, the Lustre parallel file system, or one of the /nfs/tmp*n* file systems.

Managing Graphics Libraries

Before CHAOS 3.2, the Mesa graphics library and the NVIDIA graphics library interferred with each other on LC Linux/CHAOS clusters (because the operating system allowed the NVIDIA installer to overwrite Mesa libGL files in /usr/lib64 or /usr/lib).

Starting with CHAOS 3.2 (spring, 2007), NVIDIA files are installed in /usr/nvidia instead of /usr/lib64. The X-server configuration file and the system default library search path now look for libGL in /usr/nvidia *before* checking the usual library directories. And X11 now always renders with NVIDIA rather than with Mesa.

The chart below shows the default (mixed) graphics library and default include files on LC Linux clusters running CHAOS 3.2 or later versions. It also shows how you can switch to the alternative, nondefault library or include files if either default setting is not best for your work:

	Graphics Library	**Include Files**
Default	NVIDIA libGL	Mesa include files
Change GL to Mesa	Put /usr/lib64 into LD_LIBRARY_PATH environment variable	
Change includes to NVIDIA's		Compile with -I /usr/nvidia/include g++ option

High-Performance Interconnect

Local support for a high-performance interconnect (internal network) among CHAOS cluster nodes developed in several stages.

QUADRICS/ELAN SUPPORT:
First, the Linux Project ported the Quadrics QsNet device drivers and related software from Compaq Alpha chips running Compaq's proprietary Tru64 version of UNIX to the same chips but running Red Hat Linux. As a result, QsNet under Linux not only executed reliably, but it slightly outperformed the original Tru64 version (with a maximum bandwidth of 210 Mbyte/s).

Second, the Linux Project staff shifted focus to Intel chips and locally modified Red Hat Linux for QsNet support. Quadrics, for their part, released most of their software under an open source license and concentrated their business on Linux platforms. Meanwhile, the LC collaborators modified the system kernel used locally (now called CHAOS) to once again support QsNet in three ways:

- LC added (improved) device drivers for Quadrics Elan3 and Elan4.
- LC included the Quadrics software environment to run parallel jobs across a cluster (such as libelan, a low-level library of message-passing functions).
- LC packaged with CHAOS the Quadrics MPING ping-pong test, as part of a basic MPI test suite.

The QsNet interconnect is now available on some LC CHAOS-based Linux clusters (such as Thunder, Lilac, and ALC). Besides its direct benefits, it enables other, higher-level system features, such as a scalable parallel file system (next section).

The Elan Communication Library (libelan, mentioned above) helps optimize MPI behavior on Linux clusters with the Quadrics switch. Twenty-one environment variables (most begin with the characteristic string LIBELAN_) allow you to manage the impact of this library to:

- work around application hangs caused by communication problems,
- improve code performance under Linux (CHAOS),
- handle large amounts of message-passing memory, and
- enable Elan library support for MPI debugging.

For a current list of these environment variables and their specific roles, see this (open-network only) web site:

`http://www.llnl.gov/computing/mpi/elan.html`

INFINIBAND (OPENIB) SUPPORT:
In 2003 an ASC PathForward project began to promote commercial support for a much faster interconnect called Infiniband. The national laboratories worked with industrial partners and open-source software efforts in a collaboration (partly ASC funded) called OpenIB (see www.openib.org (URL: http://www.openib.org) for background). By 2005 the first high-performance computing (HPC) release of OpenIB became available. By 2006, LC began installing clusters (the Peleton procurement, involving machines such as Atlas and Zeus (OCF), and Rhea and Minos (SCF)) that featured an Infiniband internal network.

CHAOS evloved to support this switch innovation. By May, 2007, CHAOS version 3.2 was developed specifically for such clusters and was deployed (exclusively) on them, with OpenIB support included.

Scalable Parallel File System

GOALS:

One major service goal of CHAOS is to enable a special-purpose file system with these unusual properties:

- PARALLEL--
  allows several processes to successfully read from or write to (parts of) the same file at the same time (parallel I/O), including the case where not all parts of "a file" are contiguous on a single disk.
- SCALABLE--
  can grow very large, including many disks, with no loss of functionality or performance.
- GLOBAL--
  is available to all nodes in a cluster, even a very large cluster, at the same time (and perhaps later, to nodes spread across many clusters, as with LC's NFS-mounted common home directories).
- SECURE--
  allows jobs to readily access their own files, even if distributed across devices in a file system, while protecting each job's files from interference by other jobs running at the same time (with access authentication suitable for LLNL secure networks).

PHASES:

This goal has been approached in three phases, each of which yielded some technical refinements that helped enable the next phase.

- Compaq Partnership (2000)--
  sought to port the Petal/Frangipani research file system to Compaq (Alpha-chip) Linux nodes, taking advantage of earlier QsNet work (page 10) for high-bandwidth file transport of file-system information.
- ASCI PathForward (2001)--
  collaborated with Cluster File Systems Inc. to develop an experimental open-source, distributed, object-based file system with the properties specified above, for Linux nodes that use Intel chips.
- Lustre Lite (current phase)--
  refocused the previous effort on providing a practical, scaled-down version of a POSIX-conformant parallel file system for single (that is, *not* globally mounted) clusters implemented at Livermore Computing.
  - ◊ One aspect of this work involved moving beyond the usual "redundant array of inexpensive disks" (RAID) by designing distributed file storage using a set of virtual disks or "Object Storage Targets" (OSTs) instead. Each OST is a self-managed CPU/drive combination.
  - ◊ Another aspect of this work recruited Blue Arc Corporation to build a suitable underlying "storage appliance" to implement the OST design. Lustre Lite has been deployed on LC Linux/CHAOS clusters, and even cross-mounted among multiple LC clusters. Some scalability and reliability problems persist (see below).

USAGE ADVICE:
User information for the current Lustre implementation, which compares the actual features of LC's Lustre file systems with those of IBM's GPFS, as well as points out known pitfalls involving directory names and interactions with MPI-IO, is available in the Lustre section (URL: http://www.llnl.gov/LCdocs/ioguide/index.jsp?show=s7) of the online I/O Guide for LC.

REFINEMENTS WITH CHAOS 3.2:
Deploying CHAOS 3.2 (and later) starting in May, 2007, which includes Lustre 1.4.8, addressed three serious operational problems with earlier Lustre versions--

- Page Cache Flush--
  Under CHAOS 3.2, the SLURM epilog script that always executes immediately after your job script now flushes the page cache of Lustre pages (clean and dirty) after every job. This guarantees that the next job will start with all memory available and with no interference from delayed I/O.
- Assertion Failures--
  Under CHAOS 3.2 (Lustre 1.4.8) any Lustre assertion failures on a compute node cause the node to panic and jobs to completely terminate. Previous Lustre versions allowed nodes with assertion failures to lapse into a strange, partly failed state.
- FLOCK and FCNTL--
  Under CHAOS 3.2, system calls to FLOCK(2) and FCNTL(2) to lock Lustre files always return an error. This may affect some MPI-IO and HDF5 software. Previously, separate tasks running on different clients could use FLOCK or FCNTL to simultaneously obtain exclusive locks on the same file, clearly an operational mistake.

Parallel Resource Manager (SLURM)

The primary threefold purpose of a cluster resource manager (such as LoadLeveler on LC's IBM ASC machines or the Resource Management System (RMS) from Quadrics) is to:

- Allocate nodes--
  give users access (perhaps even exclusive access) to compute nodes for some specified time range so their job(s) can run.
- Control job execution--
  provide the underlying mechanisms to start, run, cancel, and monitor the state of parallel (or serial) jobs on the nodes allocated.
- Manage contention--
  reconcile competing requests for limited resources, usually by managing a queue of pending jobs.

At LC, an adequate cluster resource manager needs to meet two *general* requirements:

- Scalable--
  It must operate well on clusters with as many as several thousand nodes, including cases where the nodes are heterogeneous (with different hardware or configuration features).
- Portable--
  It must ultimately support jobs on clusters that have different operating systems or versions, different architectures, different vendors, and different interconnect networks. Linux/CHAOS is, of course, the intended first home for this software, but in summer 2006 LC began installing the same (vendor independent) resource manager on *both* Linux/CHAOS and AIX clusters.

Any LC resource manager must also meet two *additional*, locally important, requirements:

- Compatible with LCRM--
  Since a resource manager is not a complex scheduler nor a complete batch system with across-cluster accounting and reporting features, it must support and work well within such a larger, more comprehensive job-control framework. At LC, the Livermore Computing Resource Management (LCRM) system, formerly called the Distributed Production Control System (DPCS) (URL: http://www.llnl.gov/LCdocs/dpcs), provides that metabatch framework.
- Compatible with QsNet--
  Since LC's Linux Project has already refined QsNet (page 10) as its preferred high-speed interconnect for Linux/CHAOS clusters, an adequate resource manager must also allocate Quadrics QsNet resources along with compute nodes. (A flexible resource manager will be *interconnect independent*.)

Finally, to fit well into the emerging CHAOS environment, a resource manager at LC should ideally have these two very beneficial *extra properties* as well:

- Fault Tolerant--
  Innovative scientific computing systems are often much less stable than routine business clusters, so a good local resource manager should recover well from many kinds of system failures (without terminating its workload), including failure of the node where its own control functions execute.

- Open Source--
  The software (source code) should be freely sharable under the GNU General Public License, as with other nonproprietary CHAOS components.

No commercial (or existing open source) resource manager meets all nine of these needs. So since 2001 Livermore Computing, in collaboration with Linux NetworX and Brigham Young University, has developed and refined the "Simple Linux Utility for Resource Management" (SLURM). The summary of its requirements above gives a good profile of SLURM's role and design strategy. But it says little about how SLURM actually works.

This diagram shows SLURM's architecture (from the *system* point of view):

```
SRUN    -|        -------------
         |        |           |
SCANCEL -|--------| SLURMCTLD |--------| SCONTROL
         |        |           |
SQUEUE  -|        -------------
         |              |
SINFO   -|    ---------------------
              |         |         |
         SLURMD      SLURMD       SLURMD
              (...compute nodes...)
```

At the center is SLURM's centralized work manager (SLURMCTLD) or control daemon (with a duplicate backup for reliability, not shown). Along the bottom are the SLURMD daemons residing on every compute node, each of which runs jobs locally as a remote shell. (On BlueGene/L, compute nodes can execute only a single process so the SLURMD daemon runs instead on one of the BlueGene/L "front end nodes," but it fills the same role.) User tools (left side) allocate resources and start jobs (SRUN) on SLURM-managed nodes, terminate them (SCANCEL), report *job* status (SQUEUE), and separately report current *node* and partition status (SINFO). The administrative tool SCONTROL (right side) monitors and modifies configurations and job states. These SLURM parts were tested on an LC Linux system during 2002, then deployed for public use with the release of CHAOS 1.2 across all LC Linux clusters (that had a suitable switch) in the fall of 2003.

From the *user* point of view, SRUN is the central SLURM tool. SRUN offers over 65 command-line options that you can combine to provide:

- five ways to execute your parallel jobs. These include an interactive way much like using POE on IBM/AIX machines as well as "local batch" (without LCRM) and "global batch" (with LCRM) alternatives.
- elaborate control over resource allocation. You can specify not only the total number of nodes for your job but also the CPUs/process (multithreading), processes/CPU (overcommitment), and even specific node ranges or hostnames to use or avoid, and you can bind tasks to CPUs or memory.
- fine-grained I/O management. You can separately redirect your job's input, output, and standard error to (or from) specified files on a per-job, per-step, per-node, or per-task basis.
- detailed influence on your job's working features. You can change your job's reported name, default path, debug level, imposed "constraints" (much like PSUB), the verbosity of SRUN messages about it, and the type of MPI invoked.

On CHAOS machines, jobs submitted to SLURM using SRUN (either as a stand-alone utility or executed within an LCRM script) can be monitored for progress and resource use with the SQUEUE reporting tool. SQUEUE thus fills the role for CHAOS and SLRUM that SPJSTAT fills for AIX and LoadLeveler on IBM machines. And like SPJSTAT, SQUEUE reports jobs by means of their SLURM-assigned "local" job ID rather than their LCRM JID (even if they have one). SQUEUE also lets users request customized job-status reports, in which they can specify both the job features reported (from a list of 24) and the order in which reported jobs are sorted.

Likewise, on CHAOS machines, compute resources managed by SLURM can be monitored for features or availability with the SINFO reporting tool. SINFO thus fills the role for CHAOS that LLSTATUS fills for AIX on IBM machines. Like LLSTATUS, by default SINFO reports broadly on all node partitions, but you can focus on specific nodes or node sets if you wish. And like SQUEUE, SINFO offers customization options to change not only the node properties reported but also the order or format of columns shown in SINFO output.

On BlueGene/L *only*, an additional SLURM tool called SMAP shows the topological distribution of jobs among nodes (because job geometry is important on that machine's unusual architecture).

More details on SLURM, including how its subsystems interact with each other, how users interact with SLURM, the many specialized job-control features offered by the SRUN tool, and the customization possibilities for SQUEUE, SINFO, and SMAP output, appear in the SLURM Reference Manual (URL: http://www.llnl.gov/LCdocs/slurm). In 2006, LC began replacing LoadLeveler with SLURM for resource management even on its AIX machines. For an AIX/CHAOS and LoadLeveler/SLURM cross-comparison matrix, see the "SLURM and Operating Systems" section (URL: http://www.llnl.gov/LCdocs/slurm/index.jsp?show=s2.3) of the SLURM Reference Manual.

Cluster Administration Tools

Managing a cluster of many compute nodes often involves performing the same task on every node, or comparing the status or behavior of all nodes, or partitioning the nodes using planned configuration differences and then taking account of those differences for all future software updates. As the number of nodes grows (from clusters of 10 to 100 to over 1000 nodes), the difficulty of cluster management grows too and serious inefficiencies can appear.

The first subsection below summarizes special CHAOS-supported tools for system administrators. The second subsection introduces user tools to help anyone manage "NUMA (nonuniform memory access) nodes" on some LC Linux/CHAOS clusters.

For System Administrators

Open source cluster administration tools for Linux seldom scale up to the size of the clusters now common at Livermore Computing. So the customized CHAOS environment includes several extra, locally developed tools to promote efficient administration of very large LC clusters:

ConMan provides console management.

Every node in an LC Linux cluster is accessible via a serial console connection because the console is often the only way to (re)configure the node or check error messages if networking or other serious problems occur. But of course hundreds of tightly packed clustered nodes do not have literal monitors and keyboards attached to display and manipulate their consoles. And they often reside in buildings distant from their system administration staff.

So the CHAOS environment offers ConMan, a customized console-management program designed to maintain a persistent connection to many console devices and simultaneous users remotely. ConMan now:

- supports local serial devices and remote terminal servers with the TELNET protocol.
- maps symbolic names onto physical console devices and logs output from specified consoles into a file for later review.
- connects to virtual consoles in monitor (read-only, for continuous logging), interactive (read-write, for executing commands), or broadcast (write-only) mode.
- allows "joining" with existing clients if a console is already in use, or instead "stealing" the console's privileges.
- allows scripts to execute across multiple consoles in parallel.

Recent versions of ConMan are available through the OCF web site

`http://www.llnl.gov/linux/conman`

Genders

facilitates cluster configuration management.

A standard practice among LC system administrators is to codify all changes made to a cluster in a way that allows the changes to be quickly reapplied after a fresh installation. Genders facilitates this by enabling identical scripts to perform different functions depending on their context.

Genders is a simple, static, flat-file database (plain-text file) that represents the layout of a whole cluster. A Genders file (usually in /etc/genders) contains a list of node-name/attribute-list pairs, and a copy resides on each node in the cluster. Scripts then perform Genders-file lookups to configure each of many nodes appropriately. Genders also includes an rdist distfile preprocessor to expand attribute macros, so a central repository of system files can even propagate correctly to multiple clusters using this technique.

Among the current Genders software tools are:

- NODEATT--
  a query tool that lists all nodes with a specified attribute (useful as a conditional test in scripts).
- DIST2--
  an rdist preprocessor that can quickly redistribute appropriate configuration file variations when a Genders file changes.
- CROUTE--
  a Perl script that expresses network routing schemes (for load balancing) in a single configuration file.
- C and Perl APIs--
  to query Genders files or, with the help of PDSH (below), to target a command to just those nodes that share a common Genders attribute.

Recent versions of Genders are available through the OCF web site

`http://www.llnl.gov/linux/genders`

Intelligent Platform Management Interface (IPMI)

is a standard specification, developed by Dell, HP, Intel, and NEC, for a way to remotely monitor and manage the physical status (temperature, air flow, power) of computer nodes. IPMI is implemented by hardware vendors at the chip level, so application users are often unaware of it. It relies on a "baseboard management controller" (BMC), a small processor that supports IPMI separately from each chip's main CPU and its operating system.

CHAOS system administrators use several locally developed software tools to take advantage of IPMI features to check or control nodes on LC's large Linux clusters (all are shared as open source):

BMC-WATCHDOG	runs as a daemon to manage and monitor the "baseboard management controller" (BMC) timer, which enables several system timeout functions as well as resetting after each operating system crash.
IPMIPING	implements the IPMI ping (path checking) protocol, as well as the Remote Management Control Protocol. Both are used mostly to debug IPMI over local area networks.
IPMIPOWER	works in conjunction with PowerMan (see below) to remotely control compute-node power supplies by using IPMI.

PAM (Pluggable Authentication Modules for Linux) simplifies both user authentication and the management of resource limits for Linux clusters.

Strictly speaking, PAM is a suite of shared libraries designed to provide middleware that bridges "access applications" (such as FTP, LOGIN, PASSWD, and SSH) and authentication mechanisms (Kerberos, DCE, RSA SecureID tokens). All Red Hat Linux systems come with PAM by default because this makes authentication management much easier for system administrators.

In addition, CHAOS systems that use LCRM to control batch jobs and to manage job resources have taken advantage of PAM to simplify the use of resource limits as well. Starting in February, 2006, system administrators no longer need to modify local LCRM configuration files to alter resource limits because LCRM now lets PAM manage four CHAOS limits dynamically: core size, stack size, number of open file descriptors, and number of processes allowed for any single user.

PDSH executes commands on remote hosts in parallel.

The Parallel Distributed Shell (PDSH) utility is a multithreaded remote shell client for system administrators, simmilar to IBM's DSH tool but with better error handling. PDSH offers several remote shell services including RSH, SSH, and Kerberos IV. It is designed to gracefully handle the usual kinds of node problems, such as when a target node is down or slow to respond. Using PDSH lets a system administrator:

- execute commands across all nodes of a large cluster as if it were a single machine (simple commands can execute on over 1000 nodes of a typical cluster in less than 2 seconds), and
- run small MPI jobs in parallel across the QsNet interconnect, which is helpful for trying parallel test cases or interconnect diagnostics on a new system that still lacks a regular resource manager. (page 14)

Recent versions of PDSH are available through the OCF web site

`http://www.llnl.gov/linux/pdsh`

PowerMan manages clustered-system power controllers.

Power management for a large cluster poses challenges very like those posed by console management (above). To minimize circuit loads or focus repairs, a system administrator may want to boot an entire cluster, just one rack, or even an individual node. And as with consoles, remote power control is important when the cluster resides in a different building than the staff.

PowerMan therefore provides a remote, command-line interface for a wide variety of power-control and monitoring devices, through a TCP network connection. There is no standard protocol for a power-control device interface, so PowerMan offers a flexible configuration that can adapt to almost any hardware. PowerMan can query both plug and power supply output status, and it can power-on, power-off, power-cycle, and hard-reset individual nodes or node ranges in a CHAOS cluster. Where hardware allows, PowerMan can also flag nodes needing service and gather out-of-band temperature data.

Recent versions of PowerMan are available through the OCF web site

`http://www.llnl.gov/linux/powerman`

WHATSUP quickly detects and reports which nodes are currently up and down within the Linux cluster where you run it. When executed (by any user, not just administrators) with no options, WHATSUP summarizes the count and name list of up nodes, followed by the count and name list of down nodes (then automatically ends). You can optionally report only up nodes (--up), only down nodes (--down), or node lists not summarized but instead separated by commas (--comma), returns (--newline), or blanks (--space).

YACI installs the operating system on cluster nodes.

Livermore Linux clusters currently run with a full copy of the CHAOS/Red-Hat operating system on every node. The alternative, using a root file system shared across nodes, posed concerns about the performance and reliability of network file servers (as well as complications regarding the integrity of Red Hat "packaging"). Multiple copies on multiple nodes, however, means that deploying or upgrading the operating system requires a scalable way to transfer (or "image") the operating system onto a large number of nodes. No available open source techniques (such as VA System Imager and LUI) met local needs. So the CHAOS staff developed YACI (Yet Another Cluster Installer).

A YACI installation begins by creating a disk partition and placing the needed files in a staging partition. These are then converted into compressed TAR images ("tarballs"). Cluster installation continues by installing management node(s) from a YACI CD-ROM, then network booting a stand-alone image onto the remaining nodes. Finally, each stand-alone image partitions its local disk and deploys the "tarballs" of the original disk partition. Once the management node(s) are configured, YACI can install CHAOS on the remaining 1152 nodes of LC's MCR cluster in about 50 minutes.

Recent versions of YACI are available through the OCF web site

    http://www.llnl.gov/linux/yaci

For Users of NUMA Nodes

Many of the Linux/CHAOS clusters that LC bought starting in late 2006 (such as Atlas and Zeus) have hardware with "nonuniform memory access" (sometimes called nonuniform memory architecture), always abbreviated NUMA. Starting with version 3.2, released in 2007, CHAOS includes extra features to enable users to more effectively work with NUMA hardware. This section explains basic NUMA concepts and CHAOS policy regarding NUMA resources. It then introduces (in separate subsections) three user tools to help manage those resources.

NUMA Hardware:
Linux/CHAOS clusters with NUMA hardware have addressable nodes (such as atlas36 or atlas1151) each comprised of four "dual-core sockets" (components wtih two CPUs each), as shown here:

```
             -----------------------------
addressable  | cpu0 | cpu0 | cpu0 | cpu0 |
node         | cpu1 | cpu1 | cpu1 | cpu1 |
(atlas1151)  -----------------------------
              node0 |node1 |node2 |node3  ---dual-core sockets
                                             = NUMA nodes
```

Each socket functions as a kind of subnode (here 0 through 3) with its own CPUs and local memory. But because the (two) CPUs share their memory with different access rates (that is, nonuniformly), their socket is a "NUMA node." Each addressable node on LC clusters with NUMA hardware contains four such NUMA nodes.

This diagram shows in more detail the nonuniformity of memory access within each NUMA node:

```
|-----one socket, one NUMA node-------|
-----------------     -----------------
|     CPU0      |     |     CPU1      |
|               |     |               |
-----------------     -----------------
|   I   |   D   |     |   I   |   D   |   L1 (fast)
| cache | cache |     | cache | cache |
|       |       |     |       |       |
------------------- -------------------
|    L2 cache   | | | |    L2 cache   |   L2 (slower)
|               | | | |               |
----------------  | |  ----------------
                  | |
          -------------------
          |  crossbar/bus   |             Remote memory,
          -------------------             local to other CPU
                                          (slowest)
```

For each CPU, access to its instruction (I) and data (D) L1 local memory is faster than access to its local L2 cache. And for each CPU, the other CPU's memory is also accessible (via a bus) but is "remote" or "foreign." On machines (such as LC's Thunder) where the remote-memory access rate *equals* the local-memory access rate, the hardware is called SMP (symmetric multiprocessor). On machines (such as

Atlas or Zeus) where the remote-memory access rate is *slower* than the local-memory access rate, the hardware is called NUMA.

Memory Policy:
The default memory policy under CHAOS 3.2 or later on LC machines with NUMA hardware is to allocate memory to a process from its local NUMA node. This makes it important that processes not be rescheduled to different CPUs (and certainly not to different NUMA nodes) during long-running jobs. So CHAOS now automatically enables "CPU affinity" for SRUN-launched jobs on LC machines with NUMA hardware. CPU affinity keeps a process from moving between CPUs, thus avoiding the performance overhead of transferring the process's working set between different CPU caches. The three tools described in the subsections below are available on NUMA-hardware machines to query and adjust this CHAOS memory policy and the CPU-affinity default setting.

TASKSET (Set CPU Affinity)

TASKSET lets you manage the CPU affinity of your processes on LC Linux/CHAOS machines that have NUMA hardware. The Linux scheduler honors the CPU affinity that you declare with TASKSET and will not run the specified process on any other CPUs (or you can discover a process's previously set CPU affinity). The parent (page 22) of this subsection explains the meaning of NUMA hardware and CPU affinity more fully. TASKSET can operate on a running process (if you know its PID) or you can launch a new command with a specified CPU affinity. NUMACTL (page 26) reports information helpful for planning your use of TASKSET.

SYNTAX:
TASKSET has a rather unusual execute-line syntax. To run it, type:

```
taskset options [mask|cpulist] [pid|command]
```

where (from right to left) *pid* or *command* specifies the process whose CPU affinity you want to control, *mask* or *cpulist* specifies which CPUs you want to bind to the target process (either by hexadecimal bitmask or comma-delimited list of CPU numbers, perhaps with hyphenated ranges), and *options* guide TASKSET's behavior. The available *options* are:

-p	(--pid) targets a specified running process using its PID instead of launching a new command. WARNING: the PID always comes at the right-hand end of the execute line even though the -p option comes at the left-hand end, counterintuitively *separated* by the affinity mask or list of CPUs. See the examples below.
-c	(--cpu-list) specifies CPU affinity with a comma-delimited list of CPU numbers instead of with a hexadecimal mask. (Again, the argument for -c falls *between* -p and its argument if both -p and -c are used. See the examples below.)
-h	(--help) displays a short list of available options and ends.
-V	(uppercase vee, --version) reports the current TASKSET version and ends.

EXAMPLES:
Typical examples of using TASKSET and its unusual syntax include--

(A) Report the current CPU affinity (mask) for the process with PID 10403:

```
taskset -p 10403
```

(B) Set to hexadecimal ff the CPU affinity (mask) for the process with PID 10403 (note the unusual ordering of the arguments here, with PID on the right-hand end of the excute line):

```
taskset -p ff 10403
```

(C) Set to CPUs 5 and 6 the affinity of running process 10403 (again note the unusual order of the two arguments):

```
taskset -pc 5-6 10403
```

(D) Execute the DATE command only on CPU 4:

```
taskset -c 4 date
```

NUMACTL (Control Memory Policy)

NUMACTL has both a reporting role and a role in setting memory policy on LC Linux/CHAOS machines that have NUMA hardware. The parent (page 22) of this subsection explains the meaning of NUMA hardware and memory policy.

Some NUMACTL options report on current NUMA node resources or their memory policies. Other NUMACTL options let you specify the memory policy that applies to a command that NUMACTL executes. (Still other options, omitted here, set memory policy for "shared memory segments" on machines where those exist.)

REPORTING NUMA NODES:
To discover details about the NUMA hardware on the addressable node (such as atlas32) where you execute NUMACTL, type

```
numactl --hardware|--show
```

where

--hardware lists the "NUMA nodes" (multiple-cpu sockets) on the current addressable node and reports the total and free memory for each one. A typical report has the form:

```
available: 4 nodes (0-3)
node 0 size: 3311 MB
node 0 free: 1072 MB
...
```

--show lists the memory-policy attributes that you can set by using the NUMACTL options listed below and reports the current value of each attribute. A typical report has the form:

```
policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7
cpubind: 0 1 2 3
node bind: 0 1 2 3
mem bind: 0 1 2 3
```

SETTING MEMORY POLICY:
To specify your desired memory-allocation policy for the NUMA node(s) where a specified *command* executes, type

```
numactl policyopt command [arguments]
```

where *policyopt* is one of the following mutually exclusive alternatives (in order from most restrictive to most flexible, plus two others for CPUs):

--localalloc (-l) always allocates memory only on the node where the process runs (but no others).

--membind=*nodes*

(-m *nodes*) allocates memory only on the specified *nodes* (a comma-delmited list of NUMA-node numbers, a hyphen-linked node range A-B, or all). If the specified nodes lack enough free memory, the allocation fails.

--preferred=*node*

(no short form) allocates memory "preferably" on the specified *node* (always a single node number), but uses memory from other NUMA nodes if the free memory on the chosen node is insufficient.

--interleave=*nodes*

(-i *nodes*) allocates memory round-robin from the specified *nodes* (same syntax as for --membind). When free memory is not available on any interleave target node, memory from other NUMA nodes in the round-robin set is used.

--cpunodebind=*nodes*

(-N *nodes*) executes *command* only on CPUs of the specified node(s) (same syntax as for --membind).

--physcpubind=*cpus*

(-C *cpus*) executes *command* only on CPUs specified by their physical CPU numbers as shown in the /proc/cpuinfo file.

NUMA-MAPS (Display Memory Use)

NUMA-MAPS lets you display information about how your current processes are using memory on NUMA hardware, including files, stack, and heap (or subsets that you select). The parent (page 22) of this subsection explains the meaning of NUMA hardware as well as the affinity and memory-policy issues that pertain to it.

To report on your NUMA memory use, type

```
numa-maps processoption scopeoption
```

Here *processoption* selects the process(es) to cover and can be any ONE of these mutually exclusive alternatives:

--user=*uname*	(-u *uname*) reports the PID, Unix command, current CPU mask, total memory used and memory per NUMA node for all processes owned by *uname*.
--name=*string*	(-n *string*) reports the PID, Unix command, current CPU mask, total memory used and memory per NUMA node only for the process whose name exactly matches *string* (filter characters are not allowed).
--all	(-a) reports on all processes of the user who runs NUMA-MAPS (root users will want to combine this with -q to ignore kernel threads).

Likewise, *scopeoption* selects the granularity of the report (the default scope is mapped stack and heap memory only):

--heap-only	(-H) displays NUMA memory information only for the heap.
--stack-only	(-S) displays NUMA memory information only for the stack.
--full	(-F) displays NUMA memory information for all mapped files, stack, and heap.

If you plan to pipe NUMA-MAPS output into another program for further processing you can omit the descriptive header by invoking the --no-header (-n) option along with any others. And root users will want to include --ignore-zero (-q) to skip processes (such as kernel threads) without any mapped pages.

Environment Variables For CHAOS

Just as AIX supports many environment variables unique to IBM's "Parallel Operating Environment" (POE) (URL: http://www.llnl.gov/LCdocs/poe), intended to help manage parallel jobs only on IBM machines, so too does CHAOS provide unique environment variables to manage parallel jobs on the LC Linux clusters where it is the operating system. Currently the CHAOS-only environment variables related to the Elan library (a Quadrics library of low-level message-passing functions) are in use on LC production clusters (such as Thunder, Lilac, and ALC).

Default values for CHAOS environment variables are noted if they exist, and for most jobs the default value is the optimal value. Additional Elan library environment variables, mostly related to controlling Quadrics switch support for MPI optimization or debugging, are explained at this dedicated web site (open-network only):

`http://www.llnl.gov/computing/mpi/elan.html`

The CHAOS Simple Linux Utility for Resource Management (SLURM) uses its own additional environment variables (all with names that begin with "SLURM_") to store resource-allocation values for the jobs that SLURM manages. See the "Environment Variables" section of the SLURM Reference Manual (URL: http://www.llnl.gov/LCdocs/slurm) for an explanatory list. For a big-picture look at how the CHAOS environment variables here and the SLURM environment variables both fit into the larger, complex set with which LC manages its systems and jobs, consult LC's Environment Variables user guide (URL: http://www.llnl.gov/LCdocs/ev).

LIBELAN_GALLOC_EBASE

(default value: 0xb0000000) resizes the Elan global memory heap for MPI collective operations. EBASE is a pointer to a base virtual address in Elan memory to be used for the global heap. (Set this variable and the next two if you use MPI collectives, such as REDUCE, GATHER, SCATTER, or their ALL versions, with more than about 100 processes.)

LIBELAN_GALLOC_MBASE

(default value: 0xb0000000) resizes the Elan global memory heap for MPI collective operations. MBASE is a pointer to the main memory base in Elan memory to be used for the global heap.

LIBELAN_GALLOC_SIZE

(default value: 16777216) resizes the Elan global memory heap for MPI collective operations. SIZE is the size in bytes of the Elan global heap.

LIBELAN_WAITTYPE

(suggested value: POLL) specifies how a blocking MPI process will share computing resources (comparable to MP_WAIT_MODE under POE on IBM machines). Possible values are:

POLL	(default) has the receiving thread actively poll for incoming messages. Use this choice for all MPI jobs on clusters that have a Quadrics interconnect.
SLEEP	has the receiving thread sleep and thus remove itself from the active dispatching queue.
YIELD	has the receiving thread stay in the queue but yield the processor if it has no work to do.

MALLOC_TRIM_THRESHOLD

(suggested value: -1) see the next item for joint use.

MALLOC_MMAP_MAX

(suggested value: 0) when combined, MALLOC_TRIM_THRESHOLD and MALLOC_MMAP_MAX force MALLOC to use SBRK() rather than MMAP() to allocate memory. This improves performance, but it may reduce the total amount of memory available to your user processes (to no more than 1 Gbyte/process).

MPI_USE_LIBELAN

toggles the Elan library optimizations. Possible values and their roles are:

1	(default) enables the Elan library optimizations.
0	disables the Elan library optimizations. Use this only for debugging, if you suspect problems with the Elan libraries themselves.

OMP_NUM_THREADS

controls the number of threads spawned by Intel's Math Kernel Library (MKL) routines when you invoke this threaded library on any of LC's Intel Linux machines (such as ILX, MCR, or Thunder). By default, MKL sets the number of threads equal to the number of processors.

Exec-Shield Security Feature

A security feature called "exec-shield" is enabled by default for all executables on LC clusters that run CHAOS 3.0 (or higher), which was gradually deployed late in 2005. Exec-shield prevents most (but not all) data areas from becoming executable after an attack that overwrites data structures or that tries to put malicious code into data structures. It thus protects against (most) stack, buffer, and function-pointer overflow attacks. This section summarizes how exec-shield works, its computational overhead, possible negative side effects on some application codes, and how to disable exec-shield for a specific executable to avoid those side effects.

STRATEGY:
Exec-shield relies on a limit-setting feature available on x86 (but not on Itanium, ia64) processors. Thus it works on LC Linux machines ALC (on OCF) and Lilac (on SCF), for example, but not on Thunder.

On CHAOS 3.0 (or later) machines (with x86 chips), exec-shield causes the kernel to always maintain a "maximum executable address" value or "exec-limit." With every context switch, the scheduler enforces this exec-limit for the code segment of the currently running process or thread. Each process can have a different exec-limit, which the scheduler imposes dynamically so that the appropriate limit is the one in play at any time.

In addition, with exec-shield enabled the kernel remaps all standard PROT_EXEC mappings into the x86 addresses from 0 to 16 Mbyte. This is the "ASCII-armor" area, so-called because addeses here cannot be jumped to by using ASCII-based overflows (such as extra-long URLs).

The result of these two measures (limiting the executable addresses and protecting those that are executable from ASCII-based overflows) is that

- the stack,
- the MALLOC heap, and
- most of the MMAP data area (the shared libraries)

are *not* executable. This provides extensive (though not completely foolproof) protection from overflow attacks for application codes running under CHAOS 3.0.

OVERHEAD:
Exec-shield was designed to be efficient. Only two or three cycles of overhead are lost for every PROT_MMAP system call. An additional two or three cycles are lost for every context switch that occurs with exec-shield enabled.

SIDE EFFECTS:
Application codes that assume a static layout for their program stack and heap, or that make other (previously harmless) assumptions about the specific layout of virtual address space, may destabilize and show random crashes under CHAOS 3.0 with exec-sheild enabled. This feature can also make debugging more difficult.

To allow a work-around in case such problems occur, LC has installed exec-shield at "security level 2." This means that it is enabled by default but that you can disable it on request for a specific binary file for which exec-shield is causing unintended trouble. See the next subsection for work-around instructions.

Note also that CHAOS 3.0 includes the Xorg X-window system instead of the previously used XFree86 version. This is specifically because XFree86 is incompatible with the executable-stack limitations imposed by enabling exec-shield.

DISABLING EXEC-SHIELD:
To avoid random crashes of a previously stable application code, or to allow effective debugging of a newly developed code, you can disable the CHAOS-3.0 exec-shield security feature for a specified binary file in two ways:
(1) At compile time.
Invoke the linker flag -z with the argument execstack. For example, with GCC use

```
gcc -o test -Wl,-zexecstack test.c
```

(2) At run time.
EXECSTACK is a user tool to set, clear, or query the executable status of the stack flag for "ELF binaries," which is the standard "executable and linking format" for binaries under Linux. If you have already linked a code and it really needs an executable stack (despite the security risks involved), then run

```
execstack -s pathname
```

where the argument here is the full pathname of your executable file.

Disclaimer

Keyword Index

To see an alphabetical list of keywords for this document, consult the next section (page 35).

Alphabetical List of Keywords

Keyword	Description
a	The alphabetical index of keywords.
admin-tools	Tools for system administrators.
availability	Where CHAOS runs.
chaos-features	Key components of CHAOS environment.
chaos-goals	CHAOS design goals, needs met.
cluster-administration	Cluster tools supported by CHAOS.
date	The latest changes to this document.
diskless-nodes	How CHAOS handles diskless nodes.
entire	This entire document.
environment-variables	CHAOS-relevant env. variables.
exec-shield	Exec-shield CHAOS security feature.
graphics-libraries	Managing NVIDIA, Mesa interactions.
index	The structural index of keywords.
interconnect	Support for QsNet node interconnect.
introduction	CHAOS operating system overview.
numactl	Controls NUMA memory policy.
numa-maps	Shows NUMA memory use.
numa-tools	Tools for users of NUMA nodes.
parallel-file-system	Requirements, phases of parallel I/O support.
problems-addressed	Four problems addressed by CHAOS.
resource-manager	SLURM goals, features, and user tools.
revisions	The complete revision history.
scope	Topics covered in this document.
taskset	Manages NUMA CPU affinity.
title	The name of this document.
user-tools	Tools for users of NUMA nodes.
who	Who to contact for assistance.

Date and Revisions

Revision Date	Keyword Affected	Description of Change
12Jun07	chaos-features	Subdivided and expanded.
	cluster-administration	Subdivided and expanded.
	diskless-nodes	How CHAOS 3.2 handles diskless nodes.
	graphics-libraries	NVIDIA, Mesa complex interactions.
	interconnect	Infiniband support explained.
	parallel-file-system	Lustre 1.4.8 refinements spelled out.
	numa-tools	More support for NUMA hardware added.
	taskset	NUMA affinity tool added.
	numactl	NUMA memory-policy tool added.
	numa-maps	NUMA memory-reporting tool added.
	index	New keywords for 8 new sections.
03Oct06	resource-manager	SLURM details expanded, AIX role noted.
27Mar06	cluster-administration	LCRM support for PAM added, explained.
25Jan06	environment-variables	Cross ref added to Env. Vars. manual.
21Nov05	chaos-goals	Cross ref. to exec-shield added.
	chaos-features	How to detect CHAOS version added.
	exec-shield	New section explains security feature.
	index	New keyword for new section.
03May05	resource-manager	SLURMD daemon and SMAP tool differences on BlueGene/L noted.
08Mar05	interconnect	Elan environment vars. site.
	environment-variables	Cross ref to Elan MPI vars. added.
14Feb05	parallel-file-system	Lustre usage help cross ref. added.
24Jan05	cluster-administration	IPMI and WHATSUP tools added.
10Nov04	chaos-features	Scheduler flexibility cross ref'd.
	resource-manager	SINFO roles, features spelled out.
24May04	resource-manager	SQUEUE role, features spelled out.
18Mar04	introduction	Details clarified.
	resource-manager	

		SRUN user benefits spelled out.
28Oct03	chaos-features	SLURM now deployed.
	resource-manager	SINFO added, SRUN elaborated.
	environment-variables	Cross ref to SLURM variables added.
25Aug03	chaos-goals	Cross ref to SLURM manual added.
	resource-manager	Cross ref to SLURM manual added.
15Jul03	chaos-goals	LLNL Linux web site noted.
	chaos-features	Local kernel modifications summarized.
	cluster-administration	More tool details, links added.
10Mar03	entire	First edition of CHAOS manual.

TRG 12Jun07